Chapter 1

Hello World!

Our first application displays “Hello, World!” on the device display. Its code is as follows:

```
appHelloWorld
    .txtHelloWorld
        :txtValue = Hello, World!
        :numCenterX = 50
        :numCenterY = 50
```

In Hilltop, an application is a hierarchical collection of *object definitions* provided by the programmer. When the application is run, *object instances* are created in the computer’s memory based on these object definitions.

appHelloWorld is the name of an application-type object that represents the application itself. All objects in Hilltop are based on predefined object types that are identified by a prefix[1] that is added to the object’s name. The **app** prefix identifies **appHelloWorld** as an application-type object. The application object always appears as the first or “root” object in the code listing. The application object is also a window-type object that, by default, is displayed fullscreen and has a white background, although its visibility, background color, height, width, position on the device display, and other attributes may be controlled by the programmer.

.txtHelloWorld is the name of a text-data-type object that represents the text that is to be displayed. The **txt** prefix identifies **.txtHelloWorld** as a text-data-type object. The period before the **txt** prefix indicates that **.txtHelloWorld** is a child object, in this case of the **appHelloWorld** object. Its full name is **appHelloWorld.txtHelloWorld**, although the name of the application object need not be explicitly specified when referring to its descendant objects. Child objects appear indented relative to their parent objects.

All objects in Hilltop have predefined *attributes* which vary according to object type. An object’s attributes provide information about the object. As a text-data-type object, **.txtHelloWorld** has a **:txtValue** attribute that represents a text value. Like objects, attributes have types that are identified by a prefix that is added to the attribute’s name. The **txt** prefix identifies **:txtValue** as a text-data-type attribute. The colon before the **txt** prefix indicates that **:txtValue** is an attribute. Attributes appear indented relative to their parent objects and are listed before their parent object’s child objects.

[1] The author is designing an application editor that will require keyboard input only when first naming objects, entering literal values, and entering comments. Thereafter, the programmer will indicate object and attribute names, function names, and other programming constructs via context menus. See https://youtu.be/jqoMHHefy9k for a demonstration of this concept. The application editor will automatically add object and attribute type indicators (prefixes), control indentation of objects, attributes, and code blocks, and will allow the programmer to customize code display characteristics, including whether and how object and attribute prefixes are displayed, whether curly braces, indentation, or statements (e.g., EndIf) are used to indicate code block boundaries, whether = or := is used for assignment, whether = or == is used for relational operators, etc.

Just as the programmer defines the objects that make up an application, so, too, does the programmer define the attributes that are associated with an object definition. An *attribute definition* is a set of one or more instructions that is evaluated to determine the value of an attribute, which is the data value that is stored in the computer's memory in association with an attribute. If an attribute definition is a single instruction, it may be placed on the same line in which the attribute's name appears. Thus, the value of **:txtvalue** is defined using the instruction

:txtValue = Hello, World!

When this instruction is evaluated, the literal text data value "Hello, World!" is stored in the computer's memory in association with **:txtvalue**. In Hilltop, all literal values are highlighted gray.

.txtHelloWorld also has **:numCenterX** and **:numCenterY** attributes that control where the value of its **:txtValue** attribute is displayed within its parent window object, which is the application window. The **num** prefix identifies these attributes as numeric-data-type attributes.

:numCenterX represents the center X (horizontal axis) coordinate of the **.txtHelloWorld** object and indicates where the horizontal center of the "Hello, World!" text should appear within the application window. The value of **:numCenterX** is defined as being equal to the literal numeric value 50 to indicate that the center X coordinate is located at 50% of the application window width.

:numCenterY represents the center Y (vertical axis) coordinate of the **.txtHelloWorld** object and indicates where the vertical center of the "Hello, World!" text should appear within the application window. The value of **:numCenterY** is defined as being equal to the literal numeric value 50 to indicate that the center Y coordinate is located at 50% of the application window height.

When the application is run, an instance is created of the **appHelloWorld** application object and of its child object, **.txtHelloWorld**. The text value of the **:txtValue** attribute of **.txtHelloWorld**, "Hello, World!", is displayed within the application window at the coordinates indicated by the **:numCenterX** and **:numCenterY** attributes of **.txtHelloWorld**.

As a convenience, where a data-type object represents primitive data, such as text or numeric data, the attribute definition of the attribute that holds the primitive data may be placed on the same line in which the object name appears. Thus

```
.txtHelloWorld
    :txtValue = Hello, World!
```

may be written as

.txtHelloWorld = Hello, World!

although either way, the value "Hello, World!" is stored in the computer's memory in association with the **:txtValue** attribute. Thus, the application may be rewritten as follows:

```
appHelloWorld

    .txtHelloWorld = Hello, World!
        :numCenterX = 50
        :numCenterY = 50
```

Likewise, when referring to the primitive data of a primitive data-type object, the programmer may use either the object's name (e.g., **.txtHelloWorld**) or the attribute that represents the primitive data (e.g., **.txtHelloWorld:txtValue**). We'll see an example of this later.

Maybe you're old-school and want to see a console version? Here you go:

```
appHelloWorld

    .conMyConsole
        :txtValue = Hello, World!
```

A console is a window-type object that is displayed within the application window and has a black background. The console has a **:txtValue** attribute that represents the console's text buffer. And as you may have guessed, it can be written this way as well:

```
appHelloWorld

    .conMyConsole = Hello, World!
```

Chapter 1 takeaways:

- In Hilltop, an application is a hierarchical collection of object definitions
- Objects are based on predefined object types
- An object's type is indicated by a prefix added to the object's name
- The application itself is represented by a root object in the code listing
- Child objects begin with a '.' and are indented relative to their parent objects
- Objects have predefined attributes that provide information about the object
- An attribute's type is indicated by a prefix added to the attribute's name
- Attributes begin with a ':' and are indented relative to their parent objects
- An attribute definition is a set of one or more instructions that determines the value of an attribute
- Single-instruction attribute definitions may be placed on the same line as the attribute name
- Literal values are highlighted gray
- Where a data-type object represents primitive data, the attribute definition of the attribute that represents the primitive data may be placed on the same line as the object name
- When an application is run, an instance is created of the application object and of its child objects

Chapter 2

Guessing Game

Next, let's do a guess-the-number game. Its code is as follows:

```
appGuessingGame
    .numNumberToGuess = numfunRandom(random number between 1 and 10)
    .txtPrompt = Guess a number between 1 and 10
        :numX = 30
        :numY = 30
    .numboxEntryField
        :numX = 30
        :numY =.txtPrompt:numBottomY + 5
    .txtAnswer
        :numX = 30
        :numY =.numboxEntryField:numBottomY + 5
        :txtValue
            :aplValue
                If .numboxEntryField < .numNumberToGuess
                    ParentAttribute = Too low. Guess again.
                ElseIf .numboxEntryField > .numNumberToGuess
                    ParentAttribute = Too high. Guess again.
                ElseIf .numboxEntryField = .numNumberToGuess
                    ParentAttribute = Correct! Tap the screen to play again.
                    .numboxEntryField:boolEditable = No
                    .rulePlayAgain:boolActive = Yes
                Else
                    ParentAttribute = {}
                EndIf
    .rulePlayAgain
        :boolActive = No
        :boolTriggered = Device.Display:boolTapped
            :aplWatcher
                appGuessingGame:boolInitialized = Yes
```

The application object, **appGuessingGame**, has five child objects: **.numNumberToGuess**, **.txtPrompt**, **.numboxEntryField**, **.txtAnswer**, and **.rulePlayAgain**.

.numNumberToGuess is a numeric-data-type object whose numeric value is represented by its **:numValue** attribute. Using the rule that attribute definitions of data-type objects may be placed on the same line in which their object name appears, the value of **.numNumberToGuess** is expressed as follows:

.numNumberToGuess = numfunRandom(random number between 1 and 10)

where **numfunRandom** is a predefined function that returns a number. **Random** is the name of the function, the prefix **fun** indicates that it's a function, and the prefix **num** indicates that it returns a number. The body of the function appears within parentheses and includes guide text[2] that helps the programmer understand what values are needed by the function, which in this case are the highest and lowest numeric values between which the random number is to be chosen, as well as what the function does, which is return a random number between the provided values.

.txtPrompt, is a text-data-type object that displays the instruction "Guess a number between 1 and 10" which tells the player what to do. **.numboxEntryField** is a "number box" that is a numeric-data-type object into which the user enters their guessed number. **.txtAnswer** is a text-data-type object that displays text indicating whether the player's guess is too high, too low, or correct.

The **:numX** and **:numY** attributes of these data-type objects refer to their left X and top Y coordinates within the application window. The value of the **:numX** attribute of all three of these data-type objects is defined as being equal to 30% of the width of the application window from its left edge. The value of the **:numY** attribute value of **.txtPrompt** is defined as being equal to 30% of the height of the application window from its top edge.

We can cause **.numboxEntryField** to be displayed below **.txtPrompt** by making the **:numY** coordinate of **.numboxEntryField** dependent on a vertical coordinate of **.txtPrompt** , such as on **.txtPrompt:numBottomY**. To do this, we define the value of **.numboxEntryField:numY** as follows:

```
.numboxEntryField
    …
    :numY = .txtPrompt:numBottomY + 5
```

Here, the value of the top Y coordinate of **.numboxEntryField** is defined as being equal to the value of the bottom Y coordinate of **.txtPrompt** plus an additional 5% of the height of the application window.

We can cause **.txtAnswer** to be displayed below .**numboxEntryField** in the same fashion by making the **:numY** coordinate of **.txtAnswer** dependent on .**numboxEntryField:numBottomY**.

All of the attribute definitions up to this point have been in the form of a single instruction placed on the same line in which the attribute's name appears, or, in the case of primitive data-type objects, on the same line in which their object name appears. However, if an attribute definition requires multiple instructions to determine its attribute's value, they may be placed directly into the attribute's **:aplValue** sub-attribute as follows:

[2] The application editor will automatically insert guide text within a function after the programmer specifies the function name.

```
.txtAnswer
    …
    :txtValue
        :aplValue
            If .numboxEntryField < .numNumberToGuess
                ParentAttribute = Too low. Guess again.
            ElseIf .numboxEntryField > .numNumberToGuess
                ParentAttribute = Too high. Guess again.
            ElseIf .numboxEntryField = .numNumberToGuess
                ParentAttribute = Correct! Tap the screen to play again.
                .numboxEntryField:boolEditable = No
                .rulePlayAgain:boolActive = Yes
            Else
                ParentAttribute = {}
            EndIf
```

The **apl** prefix identifies **:aplValue** as an applet-type sub-attribute[3] that holds instructions rather than a data value. The value of **.txtAnswer:txtValue** is determined by evaluating the instructions in its **:aplValue** sub-attribute. As was previously explained, when referring to the primitive data value of a primitive data-type object, the programmer may use either the object's name or the attribute that represents the primitive data. Thus, as **.numNumberToGuess** and **.numboxEntryField** are both numeric-data-type objects, they are referred to here without their **:numValue** attributes. **ParentAttribute**[4] is also used instead of **:txtValue**, as **:txtValue** is the parent attribute of the **:aplValue** attribute in which the word **ParentAttribute** appears, although **.txtAnswer:txtValue** may be explicitly specified.

An important thing to note here is that object instances are not created at runtime based on the order in which they appear in the code listing. This is because object siblings may appear in any order under their parent object. Rather, the programmer is to assume that instances of object siblings may be created *in any order* with respect to one another, subject to the following qualifications:

1. Instances of parent objects are necessarily created before instances of their child objects are created;
2. If an object B refers to an object A, an instance of object A is created before an instance of object B is created.

Thus, because the attribute definition of **.txtAnswer:txtValue** refers to **.numNumberToGuess:numValue** and **.numboxEntryField:numValue**, an instance of **.txtAnswer** is only created after instances of **.numNumberToGuess** and **.numboxEntryField** are created. Only then can the attribute definition of **.txtAnswer:txtValue** be evaluated to determine the value of **.txtAnswer:txtValue**.

---

[3] Some attributes are not programmable and their values are automatically managed, such as **:hexObjectID**, which uniquely identifies each object instance. Only programmable attributes have **:aplValue** sub-attributes.

[4] The author is undecided whether the use of **ParentAttribute** and similar constructs increases readability. In any event, the application editor will ensure that the full lineage of a currently-displayed object, attribute, and code block within the editor is always listed during scrolling, where, as in the example shown, ellipses are used to represent inermediate code.

An attribute definition of an attribute is first evaluated when an instance of the attribute's parent object is created. Thereafter, unless otherwise specified by the programmer, if the attribute definition is dependent on the value of another attribute, the attribute definition is reevaluated whenever the value of the other attribute changes. Thus, attribute values are *reactive* to values on which their attribute definitions depend. As the attribute definition of **.txtAnswer:txtValue** depends on the values of **.numNumberToGuess:numValue** and **.numboxEntryField:numValue**, the value of **.txtAnswer:txtValue** is reactive to changes in their either of their values.

The last child object of **appGuessingGame**, **.rulePlayAgain**, is a rule-type object that is triggered when a specified set of conditions is met. Its code is as follows:

```
.rulePlayAgain
    :boolActive = No
    :boolTriggered = Device.Display:boolTapped
        :aplWatcher
            appGuessingGame:boolInitialized = Yes
```

The first two attributes of **.rulePlayAgain** are **:boolActive** and **:boolTriggered**, whose **bool** prefix identifies these attributes as Boolean-data-type attributes that represent Boolean 'True' or 'False' values[5]. **:boolActive** determines whether or not the rule object is currently in effect. **:boolTriggered** controls when the rule is triggered, which occurs when the value of **:boolTriggered** is set to a Boolean 'True' value. **:boolTriggered** may receive its value from another Boolean-data-type attribute, from a function that returns a Boolean value, or from the Boolean result of a set of one or more conditions that may be specified in its **:aplValue** sub-attribute. In this example, **:boolTriggered** receives its value from **Device.Display:boolTapped**. **Device.Display** is one of a number of predefined objects[6] to which object definitions provided by the programmer may refer. The **.Display** object is a child oblect of the **Device** object, which is a root object (i.e., a sibling to the application object) representing the computing device that runs the application. **Device.Display:boolTapped** initally has no value. Whenever the display is tapped, the value of **Device.Display:boolTapped** is automatically is set to indicate a Boolean 'True' value, and is then reset to its initial state after a predefined delay.

:boolTriggered has an **:aplWatcher** sub-attribute, which is an applet-type sub-attribute that "watches" for changes in the value of its parent attribute. The **:aplWatcher** sub-attribute is available to all attributes and includes instructions that are executed whenever the value of its parent attribute changes or meets one or more values that may be optionally provided within parentheses. In this example, when the value of **.rulePlayAgain:boolActive** is 'Yes' and the display is tapped, thus triggering the rule, **.rulePlayAgain:boolTapped** is set equal to 'Yes', whereupon the instructions in its **:aplWatcher** are executed.

When the application is run, an instance is created of the **appGuessingGame** application object, as are instances of its child objects. The attribute definition of **.numNumberToGuess** is evaluated, causing a

[5] Boolean-data-type attributes may be assigned 'True' or 'Yes' values interchangeably, and 'False' or 'No' values interchangeably, and may be tested (e.g., in an **If** statement) using any of these values.

[6] Some objects are predefined, such as the computing device, its display, and its clock. Other objects are defined by the programmer.

random number to be chosen.

When the instance of **.txtAnswer** is created, the instructions in the **:aplValue** sub-attribute of its **:txtValue** attribute are evaluated. Assuming that no number has yet been entered into **.numboxEntryField**, none of the the conditions in the **If** statement comparing its value and that of **.numNumberToGuess** in the will evaluate to 'True', whereupon **.txtAnswer:txtValue** will be set equal to {} to indicate that **.txtAnswer:txtValue** has no value associated with it. {} is *not* a literal value, but represents the absence of a value.

Each time the player enters a number into **.numboxEntryField**, the value of **.numboxEntryField:numValue** changes, whereupon the attribute definition of **.txtAnswer:txtValue**, as found in its **:aplValue** attribute**,** is reevaluated, causing the value of **.txtAnswer:txtValue** to change as a result.

When the player guesses the correct number, the **:boolEditable** attribute of **.numboxEntryField** is set equal to 'No', which prevents the player from entering a new guess, and the **:boolActive** attribute of **.rulePlayAgain** is set equal to 'Yes'. When the player taps the display, **.rulePlayAgain** is triggered, **.rulePlayAgain:boolTapped** is set equal to 'Yes', and the instructions in its **:aplWatcher** are executed. The **:boolInitialized** attribute of **appGuessingGame** is set equal to 'Yes', which causes the application object, and therefore all of its child objects and their attributes, to return to their initial state when the instance of **appGuessingGame** was first created.

Chapter 2 takeaways:

- Functions are identified by the prefix **fun**
- If a function returns a value, the return value type is indicated by an additional prefix in the form ***xxx*fun**
- The body of the function may include guide text that describes the values needed by the function, as well as what the function does
- If an attribute definition requires multiple instructions to determine its attribute's value, they may be placed directly into the attribute's **:aplValue** sub-attribute
- When referring to the primitive data of a primitive data-type object, the programmer may use either the object's name alone (e.g., **.numMyNumber**) or the attribute that represents the primitive data (e.g., **.numMyNumber:numValue**)
- **ParentAttribute** refers to the attribute that is the parent of the current sub-attribute
- Object siblings (i.e., child objects that have the same parent object) may appear in any order under their parent object in the code listing
- The order in which objects appear in the code listing does not determine the order in which instances of the objects are created at runtime
- Instances of parent objects are created before instances of their child objects are created
- If an object B refers to an object A, an instance of object A is created before an instance of object B is created
- An attribute definition is first evaluated when an instance of the attribute's parent object is created
- Attribute values are reactive to values on which their attribute definitions depend
- Unless otherwise specified by the programmer, if an attribute definition is dependent on the value of another attribute, the attribute definition is reevaluated whenever the value of the other attribute changes
- An **:aplWatcher** sub-attribute is available to all attributes and includes instructions that are executed whenever the value of its parent attribute changes or meets one or more values that may be optionally provided within parentheses
- A rule-type object is triggered when its **:boolTriggered** attrubute value is 'True', whereupon the instructions in its **:aplWatcher** attribute are executed
- {} indicates that an attribute has no value associated with it. This is not a literal value, but represents the absence of a value

Chapter 3

Tic-Tac-Toe

Now for some Tic-Tac-Toe. Believe it or not, you are already familiar with most of the code in the following example! From now on, code elements that involve new concepts will be highlighted pink. Comments within the code listing will be highlighted cyan.

```
appTicTacToe
    .imgGrid
        :numCenterX = 50
        :numCenterY = 50
        :txtURL = grid.jpg
    .imgSquare<1..9>
        :txtURL
            :aplValue
                If ThisObject.txtSquareValue = X
                    ParentAttribute = x.jpg
                ElseIf ThisObject.txtSquareValue = O
                    ParentAttribute = o.jpg
                Else
                    ParentAttribute = blank.jpg
                EndIf
        :numLayer = .imgGrid:numLayer + 1
        :boolTapped
            :aplWatcher(Yes)
                Tests whether .ingSquare.txtSquareValue currently holds a value.
                If ThisObject.txtSquareValue = {}
                    ThisObject.txtSquareValue = .txtCurrentPlayer
                    .aplGameOverCheck
                EndIf
        :numCenterX
            :aplValue
                Local.numColumn = numfunModulo(remainder of ThisObject:numInstance / 1)
                Local.numWidthOfGrid = (Local.numColumn * .167) +
                                       ((Local.numColumn – 1) * .167)
                ParentAttribute = .imgGrid:numX + (Local.numWidthOfGrid * .imgGrid:numWidth)
        :numCenterY
            :aplValue
                Local.numRow = numfunInt((ThisObject:numInstance – 1) / 3)
                Local.numHeightOfGrid = (Local.numRow * .167) +
                                        ((Local.numRow – 1) * .167)
                ParentAttribute = .imgGrid:numY + (Local.numHeightOfGrid * .imgGrid:numHeight)
```

.imgSquare<1..9> (continued)

```
    .txtSquareValue

.aplGameOverCheck
    If (.imgSquare<1..3>.txtSquareValue = .txtCurrentPlayer) or
       (.imgSquare<4..6>.txtSquareValue = .txtCurrentPlayer) or
       (.imgSquare<7..9>.txtSquareValue = .txtCurrentPlayer) or
       (.imgSquare<1,4,7>.txtSquareValue = .txtCurrentPlayer) or
       (.imgSquare<2,5,8>.txtSquareValue = .txtCurrentPlayer) or
       (.imgSquare<3,6,9>.txtSquareValue = .txtCurrentPlayer) or
       (.imgSquare<1,5,9>.txtSquareValue = .txtCurrentPlayer) or
       (.imgSquare<3,5,7>.txtSquareValue = .txtCurrentPlayer)
        We have a winner!
        .imgSquare<>:boolTappable = No
        .txtGameOver = Player  + .txtCurrentPlayer +  Wins! Tap the screen to play again.
        .txtGameOver:boolVisible = Yes
        .rulePlayAgain:boolActive = Yes
    This tests whether every square has been played. ? means any.
    ElseIf .imgSquare<?>.txtSquareValue = {}
        No winner yet
        .txtCurrentPlayer = txtfunToggleValue(provide togglevalue of .txtCurrentPlayer using X
                                              and O)
    Else
        The game is a tie
        .imgSquare<>:boolTappable = No
        .txtGameOver = It's a tie! Tap the screen to play again.
        .txtGameOver:boolVisible = Yes
        .rulePlayAgain:boolActive = Yes
    EndIf

.txtCurrentPlayer = X

.txtGameOver
    :boolVisible = No
    :numCenterX = 50
    :numLayer = .imgGrid:numLayer + 2
    :numCenterY = 50
    :enuBackgroundColor =

.rulePlayAgain
    :boolActive = No
    :boolTriggered = Device.Display:boolTapped
        :aplWatcher
            appTicTacToe:boolInitialized = Yes
```

```
.ruleComputersTurn
    :boolTriggered = boolfunIf(.txtCurrentPlayer = O then Yes)
        :aplWatcher(Yes)
            Loop Local.SquareNumber from 1 to 9
                If .imgSquare<Local.SquareNumber>.txtSquareValue = {}
                    .imgSquare<Local.SquareNumber>.txtSquareValue = X
                Else
                    NextLoop
                EndIf
                If (.imgSquare<1..3>.txtSquareValue = X) or
                   (.imgSquare<4..6>.txtSquareValue = X) or
                   (.imgSquare<7..9>.txtSquareValue = X) or
                   (.imgSquare<1,4,7>.txtSquareValue = X) or
                   (.imgSquare<2,5,8>.txtSquareValue = X) or
                   (.imgSquare<3,6,9>.txtSquareValue = X) or
                   (.imgSquare<1,5,9>.txtSquareValue = X) or
                   (.imgSquare<3,5,7>.txtSquareValue = X)
                    .imgSquare<Local.SquareNumber>.txtSquareValue = O
                    .aplGameOverCheck
                    ThisObject:boolTerminated = Yes
                EndIf
                .imgSquare<Local.SquareNumber>.txtSquareValue = {}
            LoopAgain
            Loop
                Local.SquareNumber = numfunRandom(random number between 1 and 9)
                If .imgSquare<Local.SquareNumber>.txtSquareValue = {}
                    .imgSquare<Local.SquareNumber>.txtSquareValue = O
                    .aplGameOverCheck
                    ThisObject:boolTerminated = Yes
                EndIf
            LoopAgain
```

The application object, **appTicTacToe**, has the following child objects: **.imgGrid**, **.imgSquare**, **.imgSquare**, **.aplGameOverCheck**, **.txtCurrentPlayer**, **.txtGameOver0**, **.rulePlayAgain** and **.ruleComputersTurn**.

.imgGrid defines an image-type object representing the whose **:txtURL** attribute provides the location of an image file on the computing device or elsewhere on a computer network. In this example, since no path is specified, it is located where the application files are located on the device that is running the application. The image is of a Tic-Tac-Toe grid.

.imgSquare<1..9> is an image-type object definition that refers to nine specific instances of **.imgSquare** that will be created at runtime, corresponding to the nine squares of the Tic-Tac-Toe grid. The **<*n*>** suffix indicates the specific number or numbers identifying the created instance(s). Although here they are numbered 1 through 9, an object instance may be given any positive non-zero integer when the

<*n*> suffix is used. When the **<*n*>** suffix is not used in an object definition, the corresponding object instance created at runtime receives an instance number that is one greater than the currently highest-numbered instance, or 1 when it is the only instance.

The value of the **:txtURL** attribute of each object instance created using **.imgSquare** is determined by the attribute definition in its **:aplValue** sub-attribute, where **ThisObject** refers to the object instance. Depending on the value of the instance's **.txtSquareValue** child object, the value of the **ParentAttribute** of **:aplValue**, namely **:txtURL**, is set equal to "x.jpg", which is an image of the letter X, "o.jpg", which is an image of the letter O, or "blank.jpg", which is a blank image indicating that the square has not yet been played by a player.

In Hilltop, objects that are displayed are displayed in layers, where an object's layer is indicated by the value of its **:numLayer** attribute, which is 0 by default. An object may be displayed on top of or underneath another object by controlling its **:numLayer** attribute. Thus, in order to ensure that a square's X or Y image is displayed on top of the Tic-Tac-Toe grid, the value of the **:numLayer** attribute of each object instance created using **.imgSquare** is defined as being dependent on the **:numLayer** attribute of **.imgGrid** as follows:

```
.imgSquare<1..9>
  …
  :numLayer = .imgGrid:numLayer + 1
```

An **:aplWatcher** sub-attribute is defined for the **:boolTapped** attribute of each object instance created using **.imgSquare**. If the value of **:boolTapped** is 'Yes', the instructions in its **:aplWatcher** are executed to check whether the **.txtSquareValue** has a value, which indicates whether or not the square has been played. If it has not yet been played, its **.txtSquareValue** attribute is set equal to the value of **.txtCurrentPlayer**, which is either "X" or "Y". **.aplGameOverCheck** is then called to check whether or not the game has been won, where **.aplGameOverCheck** is an applet-type object that holds instructions that are executed whenever **.aplGameOverCheck** is called.

The **:numCenterX** and **:numCenterY** attributes of each instance indicate the instance's position on the Tic-Tac-Toe grid. In the **:appValue** attribute of **:numCenterX** and **:numCenterY**, the **Local** prefix denotes local variable definitions that are not visible outside of their **:aplValue** attributes.

The instructions in **.aplGameOverCheck** first check the various combinations of the **.imgSquare** object instances to determine whether or not there is a winner. Multiple instances are checked as a group using slicing, such as where **.imgSquare<1..3>.txtSquareValue** refers to the **.txtSquareValue** child objects of **.imgSquare** instance numbers 1, 2, and 3, and where **.imgSquare<1,4,7>.txtSquareValue** refers to the **.txtSquareValue** child objects of **.imgSquare** instance numbers 1, 4, and 7. If a winner is found, further moves by the players are prevented by setting the **:boolTappable** attribute of each of the **.imgSquare** instances equal to 'No' as follows:

```
.imgSquare<>:boolTappable = No
```

where **.imgSquare<>** refers to all instances of **.imgSquare**, as no specific instances are indicated within the **<>**. The value of **.txtGameOver** is set to indicate which player has won, where **+** is used to concatenate multiple text elements. **.txtGameOver** is then made visible by setting its **:boolVisible** attribute to 'Yes'.

The **:boolActive** attribute of **.rulePlayAgain** is then set equal to 'Yes', causing the game to be reset when a player taps the display.

If there is no winner, **.aplGameOverCheck** checks whether any squares remain unplayed, as follows:

ElseIf .imgSquare<?>.txtSquareValue = {}

where the **?** in **.imgSquare<?>** refers to any instance of **.imgSquare**, and thus **.imgSquare<?>.txtSquareValue = {}** tests whether **.txtSquareValue** of any instance of **.imgSquare** remains unplayed, and therefore lacks a value. If the result is 'Yes', the value of **.txtCurrentPlayer** is changed as follows:

.txtCurrentPlayer = txtfunToggleValue(provide togglevalue of .txtCurrentPlayer using X and O)

where **txtfunToggleValue** is a function that returns "X" if the value of **.txtCurrentPlayer** is currently "O", and vice versa.

If all squares have been played the game is a tie, further moves by the players are prevented by setting the **:boolTappable** attribute of each of the **.imgSquare** instances equal to 'No', the value of **.txtGameOver** is set to indicate a tie game, and **.txtGameOver** is made visible by setting its **:boolVisible** attribute to 'Yes'. The **:boolActive** attribute of **.rulePlayAgain** is then set equal to 'Yes'.

.ruleComputersTurn is a rule-type object that allows a human player ("X") to play against the computer ("O") and is triggered when it's the computer's turn to move. When **.ruleComputersTurn** is triggered, a **Loop** advances the value of **Local.SquareNumber** one number at a time from 1 to 9, in order to test each square position. If the square has already been played, **Local.SquareNumber** is advanced to the next number using **NextLoop**. If the square is unplayed, the value of the square is set to "X" and checked to see if doing so would result in the computer losing the game. If it would, the computer plays the square by setting its value to "O",**.aplGameOverCheck** is called, and **.ruleComputersTurn** is terminated by setting its **:boolTerminated** attribute to "Yes". If no such potentially winning "X" square is found, another loop is used to randomly select an unplayed square, which the computter then plays by setting its value to "O". Note that without **.ruleComputersTurn**, the game is to be played by two human players.

When the game is over, **.txtGameOver** is displayed two layers above **.imgGrid**, and its background color is set to white by setting its **:enuBackgroundColor** attribute accordingly, where **:enuBackgroundColor** is an enumeration-type attribute whose predefined values are the colors available on the computing device. **.rulePlayAgain** is triggered when the display is tapped, causing the game to be reset.

Chapter 3 takeaways:

- Specific, numbered object instances may be created by providing an object definition in the form ***.objectname<n>*** (or ***.objectname<n..m>*** or ***.objectname<n,m,…>***), where ***n*** (and ***m***, etc.) is a positive integer >=1
- When the ***<n>*** suffix is not used in an object definition, the corresponding object instance created at runtime receives an instance number that is one greater than the currently highest-numbered instance, which is 1 when it is the only instance
- Objects that are displayed are displayed in layers, where an object's layer is indicated by the value of its **:numLayer** attribute, which is 0 by default

Chapter 4

Flappy Bird

Here's Flappy Bird in under 100 lines!

```
appFlappyBird
    :enuOrientation = Landscape
    .imgBackground
        :txtURL = background.jpg
        :numX = 0
        :numBottomY = 100
        :numWidth = 100
        :numHeight = 100
        :numLayer = -1
    .imgFlappy
        :txtURL = flappy.jpg
        :numX = 20
        :numBottomY = 50
            :aplWatcher
                If ParentAttribute = 100
                    .aplGameOver
                EndIf
        :numGravity = numfunIf(.enuGameState = Playing then 3 else 0)
        :boolCollided
            :aplWatcher(Yes)
                .aplGameOver
        :boolInitialized = boolfunIf(.enuGameState = Playing then Yes)
    .vgpObstacleGroup<Template>
        :boolDestroyed = boolfunIf(ThisObject.imgTopColumn:numRightX < 0 then Yes)
        .boolFlappyAvoidedMe = No
        .imgTopColumn
            :txtURL = topcolumn.jpg
            :numX = 101
            :numBottomY = numfunRandom(random number between 10 and 30)
            :numSpeed = 4
            :numHeading = 270
```

.vgpObstacleGroup<Template> (continued)

```
    .imgBottomColumn
        :txtURL = bottomcolumn.jpg
        :numX = 101
        :numSpeed = 4
        :numHeading = 270
        :numX
            :aplWatcher
                If (ThisObject:numX < 45) and
                  (.vgpObstacleGroup<Meta>:numInstances = 1)
                    Create .vgpObstacleGroup
                    EndCreate
                EndIf
                If (ParentObject. boolFlappyAvoidedMe = No)
                    If (.imgFlappy:numX > ThisObject:numRightX)
                        ParentObject.boolFlappyAvoidedMe = Yes
                        .numScore += 1
                    EndIf
                EndIf
        :numY = ParentObject.imgTopColumn:numBottomY + 30

.enuGameState = Startup

.txtTapToPlay  = Tap the screen to begin
    :numCenterX = 50
    :numCenterY = 50
    :numLayer = 1

    .ruleSelfDestruct
        :boolTriggered = Device.Display:boolTapped
            :aplWatcher
                ParentObject:boolDestroyed = Yes

.ruleTapHandler
    :boolTriggered = Device.Display:boolTapped
        :aplWatcher(Yes)
            If (.enuGameState = (Startup or GameOver))
                .vgpObstacleGroup<>:boolDestroyed = Yes
                Create .vgpObstacleGroup
                EndCreate
                .enuGameState = Playing
            Else
                funAccelerate(.imgFlappy at 3 units/second² for .5 seconds heading 0)
            EndIf

.numScore = 0
    :boolInitialized = boolfunIf(.enuGameState = Playing then Yes)
```

```
.numBestScore = 0

.aplGameOver
    .vgpObstacleGroup<>:numSpeed = 0
    .imgFlappy:numSpeed = 0
    If .numScore > .numBestScore
        .numBestScore = .numScore
    EndIf
    .enuGameState = GameOver

.vgpGameOverText
    :boolVisible = boolfunIf(.enuGameState = GameOver then Yes)
    :numLayer = 1
    :numCenterX = 50

    .txtGameOver = Game Over
        :numCenterY = 30

    .txtScore = Score:  + txtfunNumToText(.numScore)
        :numCenterY = 40

    .txtBestScore = Best Score:  + txtfunNumtoText(.numBestScore)
        :numCenterY = 50

    .txtTapToReplay = Tap the screen to play again
        :numCenterY = 60
```

The application object, **appFlappyBird**, has the following child objects: ???.

The **:enuOrientation** attribute of the **appFlappyBird** application object is an enumeration-type attribute that controls the display orientation of the application on the device.

```
:numSpeed = 0
:numHeading = 0
:numGravity = numfunIf(.enuGameState = Playing then -0.061 else 0)
```

The **:boolCollided** attribute indicates when the **.imgFlappy** instance collides with another displayed object. **.imgFlappy:boolCollided** initially has no value. The value of **.imgFlappy:boolCollided** is automatically is set to indicate a Boolean 'True' value when a collision is detected, and is then reset to its initial state after a predefined delay. Whenever the value of **.imgFlappy:boolCollided** attribute changes, the instructions in its **:aplWatcher** attribute are executed, whereupon the **.aplGameOver** applet-type object is called and its instructions executed.

.vgpObstacleGroup<Template> is an object template definition based on which one or more object instances will be created at runtime. An object template definition may be used when it is not known at design time how many instances of an object are to be created at runtime, or when the instances are to be created programmatically at runtime, the **<Template>** may be used. When an object instance is created

using an object template, the attributes defined for the object template are copied and applied to the created object instance, and instances of any child objects defined for the object template are created as well.

[TO BE COMPLETED]

Chapter 4 takeaways:

- The **:boolCollided** attribute of a displayed object indicates when the displayed object collides with another displayed object in the same layer
- An object definition may be in the form of a ***.objectname*<Template>** from which one or more object instances may be created
- An object instance that is created using an object template receives a copy of the attribute definitions and child objects defined for the object template
- Meta-type objects, referenced in the form ***.objectname*<Meta>**, are inherently defined for all object definitions and hold information about object instances as a group